



UAEM

Universidad Autónoma
del Estado de México



Manual de prácticas de laboratorio

Título: **Análisis de Algoritmos de Ordenamiento y Búsqueda**

Para la Unidad de Aprendizaje: **Programación Avanzada**

Créditos de la Unidad de Aprendizaje: **9**

Espacio académico en que se imparte la UA: CU Valle de Chalco

Versión 1. Agosto 2017

Datos de Identificación

Programa Educativo: Ingeniería en Computación

Programa de Estudios por Competencias :Programación Avanzada

Unidad de Competencia 2: Análisis de Algoritmos de Ordenamiento y Búsqueda

Subtemas

Orden de complejidad de un algoritmo

Método de la burbuja

Método de Selección

Método de Inserción

Método de Ordenamiento rápido

Método de mezcla

Búsqueda Secuencial

Búsqueda binaria

Elaborado por:

Autor: M enT.E. Marisol Hernández Hernández

Coautor: M. en T.I. Rodolfo Melgarejo Salgado

Coautor: M. en C. Francisco Raúl Salvador Ginez

Agosto del 2017

ÍNDICE

Datos de Identificación	1
Presentación	3
Estructura de la Unidad de Aprendizaje	3
Propósito de la Unidad de Aprendizaje	4
Prácticas:	
1. Orden de complejidad de un algoritmo $O(n)$	5
2. Orden de complejidad de un algoritmo $O(n^2)$	11
3. Método de la burbuja	18
4. Método de Selección	23
5. Método de Inserción	28
6. Método de Ordenamiento rápido	33
7. Método de mezcla	38
8. Búsqueda Secuencial	43
9. Búsqueda binaria	48
Referencias	53

PRESENTACIÓN

Una vez adquiridas las habilidades de programación básicas bajo el paradigma estructurado, el alumno debe conocer otros paradigmas como la programación modular y la programación recursiva. Junto con estos paradigmas el alumno se adentra en cuestiones de algorítmica, con temas de análisis y diseño de algoritmos: funcionamiento y orden de complejidad de los métodos de ordenamiento y búsqueda y otras técnicas de diseño. Esta formación permitirá al futuro ingeniero enfrentarse a retos de programación de alta complejidad con la certeza de poder no sólo dar una solución a un problema dado sino de dar la solución óptima y ser capaz de evaluar que tan buena es la solución dada. La estructura de este manual de prácticas consta de 9 prácticas concernientes a la unidad II, cuyo nombre es: análisis de algoritmos de ordenamiento y búsqueda, cada una promueve los conocimientos necesarios para que funcione como complemento en la adquisición de las competencias que deberá tener el alumnos al término de estudiar programación avanzada y más específicamente la parte 2 de dicha unidad de aprendizaje. En cada práctica se muestran los elementos necesarios para comprender teóricamente y llevar a cabo cada método de programación, además, se agregan ejercicios y preguntas de reforzamiento como conclusión de cada aprendizaje.

ESTRUCTURA DE LA UNIDAD DE APRENDIZAJE

1. Paradigmas de programación modular y recursiva.
2. Análisis de algoritmos de ordenamiento y búsqueda
3. Diseño de algoritmos empleando diversas técnicas.



UNIDAD DE COMPETENCIA II	ELEMENTOS DE COMPETENCIA		
	CONOCIMIENTOS	HABILIDADES	ACTITUDES/ VALORES
Análisis de algoritmos de ordenamiento y búsqueda	<ul style="list-style-type: none"> - Orden de complejidad de un algoritmo - Método de la burbuja - Método de Selección - Método de Inserción - Método de Ordenamiento rápido - Método de mezcla - Búsqueda Secuencial - Búsqueda binaria 	<ul style="list-style-type: none"> • Conocer el funcionamiento de un método de ordenamiento y búsqueda • Obtener el orden de complejidad de un algoritmo de ordenamiento y búsqueda. 	Receptiva Analítica Responsabilidad para cumplir con las tareas asignadas Tolerancia y participación Desarrollar la capacidad analítica ante nuevos problemas Respetar al docente y a los compañeros mediante un comportamiento socialmente aceptable
ESTRATEGIAS DIDÁCTICAS: <ul style="list-style-type: none"> - Presentaciones acompañadas de apuntes preparados por el profesor. - Revisión y análisis de material bibliográfico - Solución de ejercicios - Desarrollo de prácticas de programación en laboratorio 		RECURSOS REQUERIDOS Pizarrón, Libro de texto y apuntes del docente Laboratorio de prácticas con un lenguaje de programación Videoprojector	TIEMPO DESTINADO 18 horas teóricas 18 horas práctica
CRITERIOS DE DESEMPEÑO II	EVIDENCIAS		
	DESEMPEÑO	PRODUCTOS	
Resolución de problemas	Ejercicios teóricos	Funcionamiento y órdenes de complejidad de los métodos de	

PROPÓSITO

Servir de enlace entre el aprendizaje de los paradigmas estructurado y orientado a objetos, a través de la programación modular. Presentar al alumno técnicas de programación avanzada como la recursividad. Proporcionar las habilidades necesarias para evaluar la complejidad de un algoritmo de ordenamiento o de búsqueda, así como estrategias para resolver problemas de alta complejidad, mediante técnicas de diseño avanzadas.

Unidad de competencia II.

Análisis de algoritmos de ordenamiento y búsqueda

Práctica 1: Orden de complejidad de un algoritmo $O(n)$

Introducción

La complejidad algorítmica representa la cantidad de recursos (temporales) que necesita un algoritmo para resolver un problema y por tanto permite determinar la eficiencia de dicho algoritmo.

Estos recursos pueden ser traducidos en tiempo y memoria. El tiempo se calcula por el coste del tamaño de los datos, tomando en cuenta el esperado, el promedio y el mejor. Si el tamaño de los datos es grande lo que importa es el comportamiento asintótico de la eficiencia.

Los ordenes de complejidad más comunes son:

Orden	Nombre	Comentario
$O(1)$	Constante	Se aplica a los algoritmos cuya ejecución se realice en un tiempo constante.
$O(\log n)$	Logarítmico	Están considerados los que implican bucles con menos iteraciones, como por ejemplo una búsqueda binaria.
$O(n)$	lineal	En este tipo de notación el tiempo crece linealmente con respecto a las iteraciones
$O(n \cdot \log(n))$	n por logaritmo de n	La mayor parte de los algoritmos tienen un orden superior, combina el logarítmico con el lineal.
$O(n^c)$, con $c > 1$	poli nómico	Aquí están muchos de los algoritmos más comunes. Cuando c es 2 se le llama cuadrático , cuando es 3 se le llama cúbico , y en general es poli nómico.
$O(c^n)$, con $c > 1$	exponencial	Aunque pudiera no parecerlo, es mucho peor que el anterior. Crece muchísimo más rápidamente.

Orden	Nombre	Comentario
O(n!)	factorial	Se prueban todas las combinaciones posibles.

Descripción de la práctica

Hacer un programa que verifique la complejidad algorítmica. Este programa calculará si un número es par o impar y dependiendo del procedimiento se aplicará la complejidad algorítmica.

Duración: 2 horas

Objetivos de aprendizaje: EL alumno aprenderá la aplicación de la complejidad algorítmica en diferentes tipos de programas.

Requisitos Previos

Tener conocimientos sobre algún lenguaje de programación, aunque estas prácticas se realizarán en lenguaje Java.

Materiales

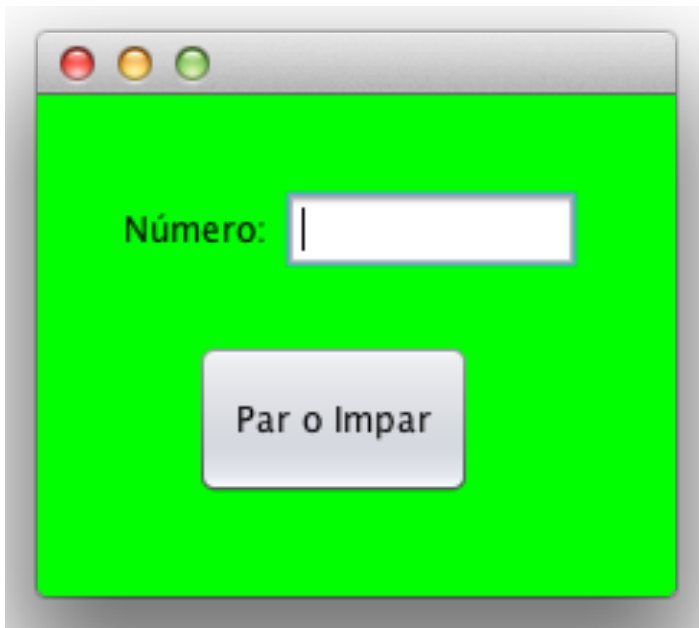
Computadora

Compilador para Java

NetBeans

Procedimiento

- 1.- Se ejecuta el programa **NetBeans**
- 2.- Se crea un nuevo proyecto cuyo nombre es par o impar y el cual determinará precisamente eso.
- 3.- Crear un formulario con los elementos (cuadro de texto, botón y etiqueta). Similar a la siguiente figura.



4.- Se codifica según el código del Container en el primer apartado, es decir en el área de declaración de variables y el constructor. El procedimiento en donde se codifica el cálculo del número par o impar se hace dando clic en el botón del formulario 'Par o Impar'.. Ver la siguiente figura:

```

package hola;
import java.awt.Container;
import java.awt.Color;
import javax.swing.JOptionPane;
public class Par_Impar extends javax.swing.JFrame {
private Object JOptionPane;
    public Par_Impar() {
        initComponents();
        Container contenedor=getContentPane();
        contenedor.setBackground(Color.blue);
    }
    @SuppressWarnings("unchecked")

```

Generated Code

```

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

    int num1;
    num1=Integer.parseInt(T1.getText());
    if(num1%2==0)
        JOptionPane.showMessageDialog(this,"Es par");
    else
        JOptionPane.showMessageDialog(this,"Es impar");
}

```

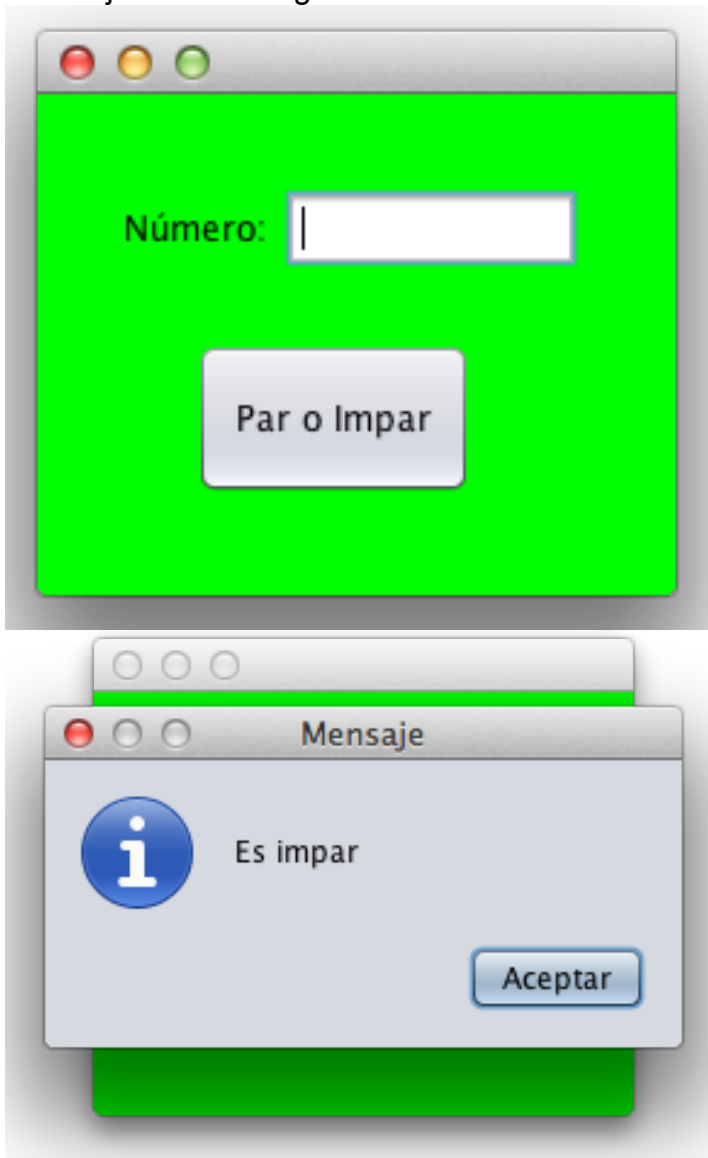
```

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    Look and feel setting code (optional)
    //</editor-fold>

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Par_Impar().setVisible(true);
        }
    });
}

```


5. Se ejecuta el código.



6. Se aplica la complejidad al procedimiento principal, específicamente al que calcula si el número es par o impar y se ve que la complejidad es $O(1)$, ya que no hace muchas iteraciones. En este caso solamente se analizará el tipo de complejidad.

Actividades de Evaluación

1. Anote en la línea la respuesta correcta: que modificación se le deberá hacer al código para que muestre verdadero el número impar.
2. **Contenedor.setBackground** indica el color del formulario verde. Anote la modificación si se deseara que el formulario fuera rosa

3. Anote cual sería la complejidad algorítmica del siguiente código:

```
public int calculoSuma(n)
{
    Int suma=1;
    for (int i=1;i<n; i++)
    { suma=suma+i; }
    return suma;
}
```

La complejidad algorítmica es: _____

Unidad de competencia II.

Análisis de algoritmos de ordenamiento y búsqueda

Práctica 2: Orden de complejidad de un algoritmo $O(n^2)$

Introducción.

La complejidad algorítmica representa la cantidad de recursos (temporales) que necesita un algoritmo para resolver un problema y por tanto permite determinar la eficiencia de dicho algoritmo.

Estos recursos pueden ser traducidos en tiempo y memoria. El tiempo se calcula por el coste del tamaño de los datos, tomando en cuenta el esperado, el promedio y el mejor. Si el tamaño de los datos es grande lo que importa es el comportamiento asintótico de la eficiencia.

Los ordenes de complejidad más comunes son:

La complejidad algorítmica representa la cantidad de recursos (temporales) que necesita un algoritmo para resolver un problema y por tanto permite determinar la eficiencia de dicho algoritmo.

Estos recursos pueden ser traducidos en tiempo y memoria. El tiempo se calcula por el coste del tamaño de los datos, tomando en cuenta el esperado, el promedio y el mejor. Si el tamaño de los datos es grande lo que importa es el comportamiento asintótico de la eficiencia.

Los ordenes de complejidad más comunes son:

Orden	Nombre	Comentario
$O(1)$	Constante	Se aplica a los algoritmos cuya ejecución se realice en un tiempo constante.
$O(\log n)$	Logarítmico	Están considerados los que implican bucles con menos iteraciones, como por ejemplo una búsqueda binaria.
$O(n)$	lineal	En este tipo de notación el tiempo crece linealmente con respecto a las iteraciones
$O(n \cdot \log(n))$	n por logaritmo	La mayor parte de los algoritmos tienen un orden superior,

Orden	Nombre	Comentario
	de n	combina el logarítmico con el lineal.
$O(n^c)$, con $c > 1$	poli nómico	Aquí están muchos de los algoritmos más comunes. Cuando c es 2 se le llama cuadrático , cuando es 3 se le llama cúbico , y en general es poli nómico.
$O(c^n)$, con $c > 1$	exponencial	Aunque pudiera no parecerlo, es mucho peor que el anterior. Crece muchísimo más rápidamente.
$O(n!)$	factorial	Se prueban todas las combinaciones posibles.
$O(n^n)$	combinatorio	intratable

Descripción de la práctica

Hacer un programa al se verifique la complejidad algorítmica. Este programa captura números en una matriz e imprime la matriz. Se le aplica el tipo de complejidad.

Duración: 2 horas

Objetivos de aprendizaje: EL alumno aprenderá la aplicación de la complejidad algorítmica en diferentes tipos de programas.

Requisitos Previos

Tener conocimientos sobre algún lenguaje de programación y de matrices, aunque estas prácticas se realizarán en lenguaje Java.

Materiales

Computadora

Compilador para Java

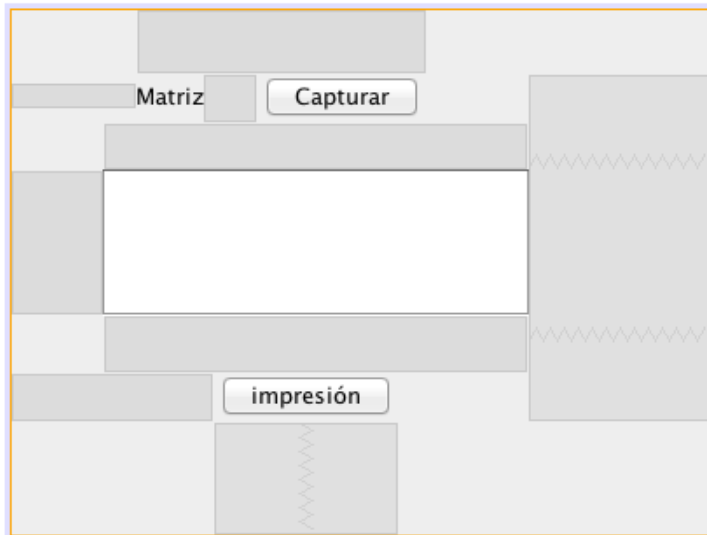
NetBeans

Procedimiento

1.- Se ejecuta el programa NetBeans

2.- Se crea un nuevo proyecto cuyo nombre será matrices, en el que se creará una matriz de 2 por 2, se capturan los datos y se imprimirá la matriz. Después se analizará para aplicar la complejidad algorítmica.

3. Se crea un formulario con los siguientes elementos una etiqueta, un área de texto y 2 botones, tal como se muestra en la figura:

The image shows a Java Swing window with a light gray background. At the top, there is a title bar. Below it, on the left, is a label 'Matriz' in a standard font. To the right of the label is a text area with a white background and a thin gray border. Below the text area is a button labeled 'Capturar' with a light gray background and a thin gray border. At the bottom of the window, centered, is another button labeled 'impresión' with a light gray background and a thin gray border. The window has a thin orange border.

4. Se codifica el siguiente código, en donde se define a la MatrizA, con dimensiones de 2 renglones por 3 columnas

```

import javax.swing.JOptionPane;

/**
 *
 * @author multimediaD
 */
public class matrices extends javax.swing.JFrame {
    int MatrizA[][]=new int[2][3];

    /**
     * Creates new form matrices
     */
    public matrices() {
        initComponents();
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    Generated Code

```

5. Se escribe el siguiente código dando clic en el botón de capturar, en donde se captura la matriz mediante un cuadro de diálogo (JOption)

```

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    for(int i=0;i<2;i++){
        for (int j=0;j<3;j++){
            MatrizA[i][j]= Integer.parseInt( JOptionPane.showInputDialog(this,"Ingresa elemento del renglón "+ i+ " columna " + j ));
        }
    }

    // TODO add your handling code here:
}

```

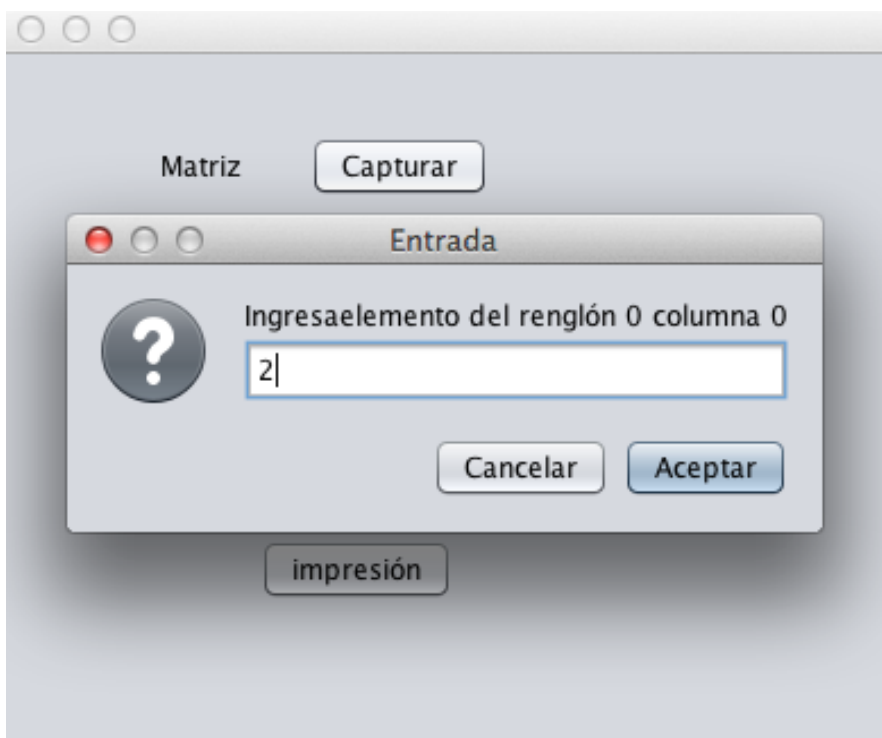
6. Se codifica el siguiente código en el botón impresión para mostrar en el área de salida la impresión de la matriz.

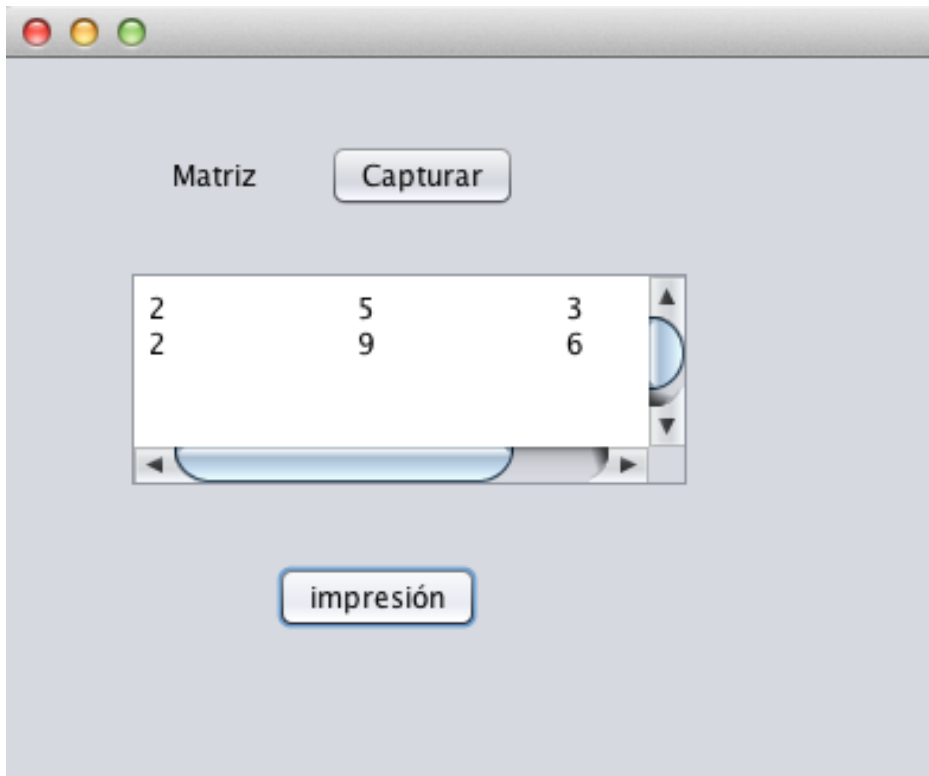
```

private void impresiónActionPerformed(java.awt.event.ActionEvent evt) {
for(int i=0;i<2;i++){
    for (int j=0;j<3;j++){
        salida.append(""+MatrizA[i][j]+"");
    }
    salida.append("\n");
}
// TODO add your handling code here:
}

```

7. Se ejecuta el programa.





8. Aplicando la complejidad algorítmica que se muestra en la introducción de este apartado, se observa que contiene 2 bucles y tomando en cuenta que la complejidad lineal crece linealmente con respecto a las iteraciones por lo tanto la complejidad se convierte a $O(n^2)$.

Actividades de Evaluación

2. Cual sería la complejidad algorítmica del siguiente código:

```
public int calculoSuma(n)
{
    Int suma=1;
    for (int i=1;i<n; i++)
    for (int x=1;x<3;x++)
    { suma=suma*x; }
    return suma;
}
```

La complejidad algorítmica es: _____

2. Tomando en cuenta el ejemplo de la práctica “matrices”, anote cual sería la complejidad algorítmica del código que imprima una matriz de 3 X 3.

3. Cual sería la complejidad algorítmica del siguiente código:


```
public int calculoSuma(n)
{
    Int suma=1;
    for (int i=1;i<n; i++)
    for (int x=1;x<3;x++)
    { suma=suma+x; }
    return suma;
}
```

Unidad de competencia II.

Análisis de algoritmos de ordenamiento y búsqueda

Práctica 3: Método de la burbuja.

Introducción.

La ordenación por el método de la Burbuja que en inglés se denomina Bubble Sort, es un algoritmo de ordenamiento muy sencillo que funciona revisando cada elemento de la lista que se está ordenando e intercambiándolo con otra posición, de tal manera que el más grande o pequeño según sea el orden va subiendo, de ahí su nombre.

Este tipo de ordenamiento es fácil de entender, sin embargo no es muy eficiente, ya que no es factible que sea aplicado a grandes cantidades de datos.

Descripción de la práctica

Hacer un programa que capture una arreglo, lo ordene por el método de la burbuja y lo imprima ordenado y desordenadamente. También se le aplicará la complejidad de Algoritmos

Duración: 2 horas

Objetivos de aprendizaje: EL alumno aprenderá la manera en que funciona el ordenamiento de burbuja

Requisitos Previos

Tener conocimientos sobre algún lenguaje de programación, de arreglos y de ordenación

Materiales

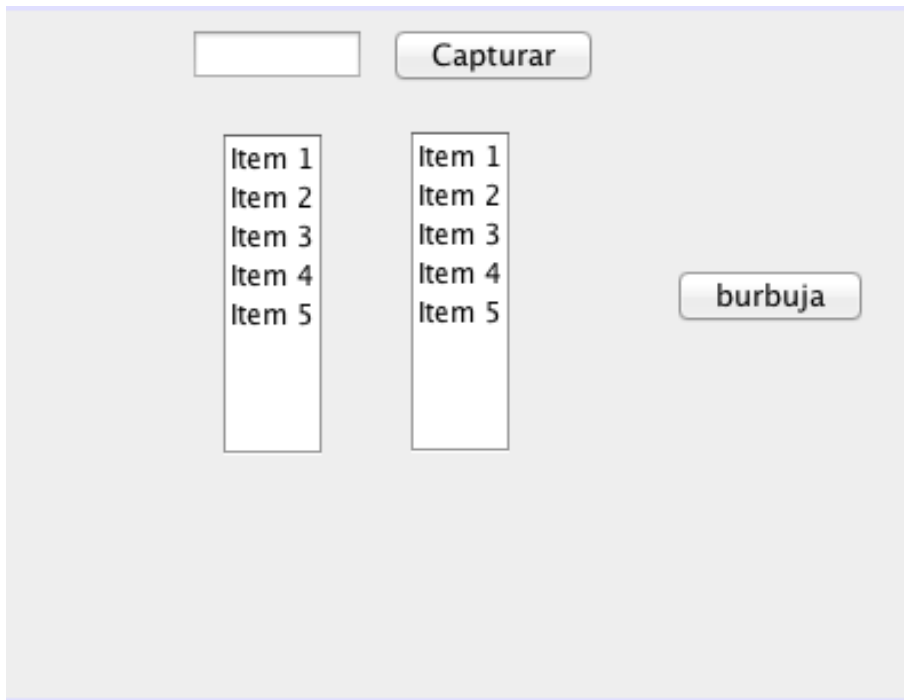
Computadora

Compilador para Java

NetBeans

Procedimiento

- 1.- Se ejecuta el programa NetBeans
- 2.- Se crea un nuevo proyecto cuyo nombre será Burbuja. Se agregan los siguientes elementos:
un cuadro de texto, 2 listas y 2 botones. Según la figura siguiente:



4. Se codifica la declaración de variables

```

L  /**
   package arreglos;

  import javax.swing.JOptionPane;

  /**
   *
   * @author multimediaD
   */
  public class burbuja1 extends javax.swing.JFrame {
    int A[]=new int[5];
    String AA[]=new String[5];
    int i=0;
    int n=5;

    /**
     * Creates new form burbuja
     */
    public burbuja1() {
        initComponents();
    }
  }

```

3. se codifica la captura del arreglo dando clic en el botón capturar. Con este botón se captura y al mismo tiempo se muestran los datos introducidos en la primera lista

```

private void capturarActionPerformed(java.awt.event.ActionEvent evt) {
    A[i]=Integer.parseInt(T1.getText());
    AA[i]=""+A[i];
    L1.setListData(AA);
    i++;
    //n=i;
    if(i==5)
        JOptionPane.showMessageDialog(this,"fin de captura");
    else{
        T1.setText("");
        T1.requestFocus();}

    // TODO add your handling code here:
}

```

5. Se codifica el botón de ordenamiento por burbuja, dando clic en el botón burbuja, el cual mostrará la segunda lista ordenada. Ese botón llamará a la función de burbuja (se visualiza más abajo) enviándole como parámetro el arreglo A y recibiendo el arreglo ordenado.

```

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

    int B[]=new int[5];
    A=burbuja(A);
    for (int i=0;i<5;i++)
    {
        AA[i]=""+A[i];
    }
    L2.setListData(AA);

    // TODO add your handling code here:
}

```

```

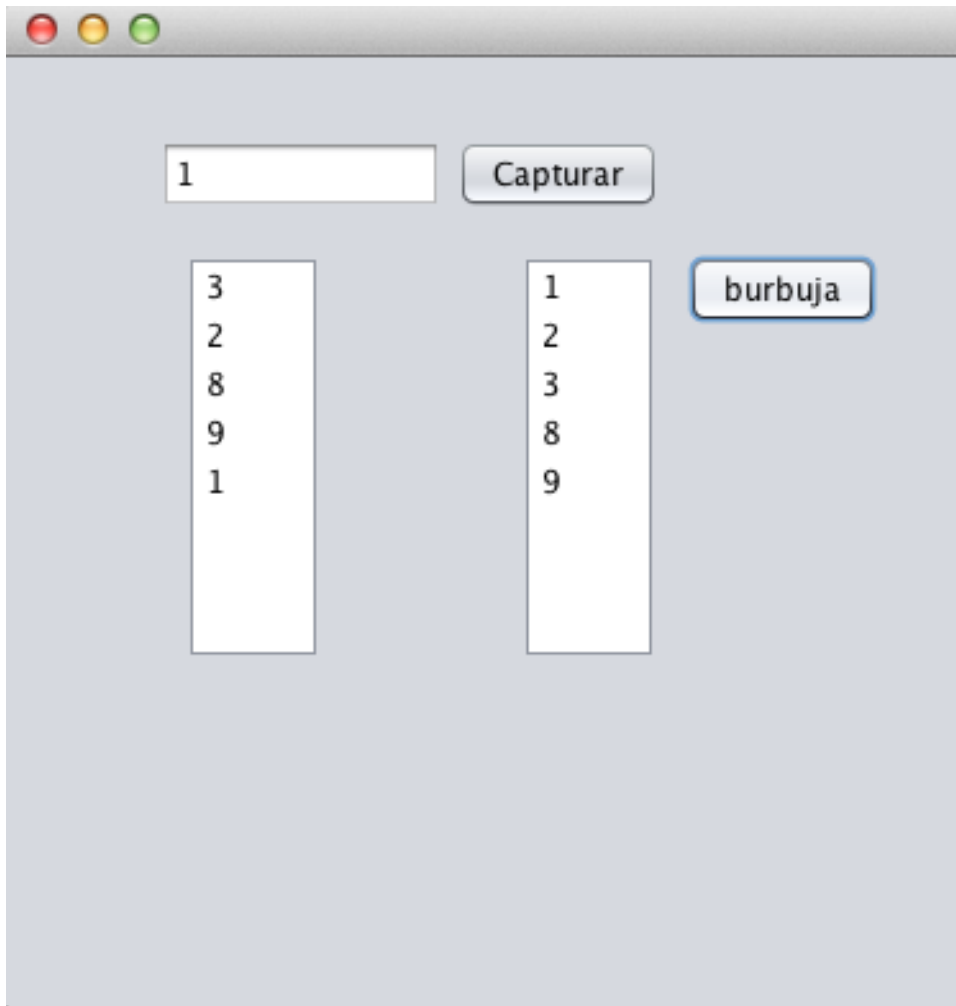
public int[] burbuja(int arreglo[])
{
    int p, j;
    int aux;
    for (p = 1; p < A.length; p++){ // desde el segundo elemento hasta
        aux = A[p]; // el final, guardamos el elemento y
        j = p - 1; // empezamos a comprobar con el anterior
        while ((j >= 0) && (aux < A[j])){ // mientras queden posiciones y el
            // valor de aux sea menor que los
            A[j + 1] = A[j]; // de la izquierda, se desplaza a
            j--; // la derecha
        }
        A[j + 1] = aux; // colocamos aux en su sitio
    }

    return arreglo;
}

```

6. Se ejecuta el procedimiento





7. Aplicando la complejidad algorítmica se observa que contiene 2 bucles y tomando en cuenta que la complejidad lineal crece linealmente con respecto a las iteraciones por lo tanto la complejidad se convierte a $O(n^2)$.

Actividades de Evaluación

1. Escribe un programa que realice capture nombres y los ordene en orden de A—Z (utilice el método de burbuja).
2. Anota el método para que se imprima un arreglo de nombres en orden descendente, es decir de la Z a la A (utilice el método de Burbuja).
3. Anota su complejidad algorítmica.

Unidad de competencia II.

Análisis de algoritmos de ordenamiento y búsqueda.

Práctica 4: Método de Selección

Introducción

El **ordenamiento por selección** que en ingles se dice **Selection Sort** es un algoritmo de ordenamiento que busca el mínimo elemento de la lista, lo intercambia con el primero, buscar el siguiente mínimo en el resto de la lista y lo intercambiarlo con el segundo, hasta llegar al final de la lista.

Descripción de la práctica

Hacer un programa que capture una arreglo, lo ordene por el método de selección y lo imprima ordenado y desordenadamente. También se le aplicará la complejidad de Algoritmos.

Duración: 2 horas

Objetivos de aprendizaje: EL alumno aprenderá la manera en que funciona el ordenamiento de selección.

Requisitos Previos

Tener conocimientos sobre algún lenguaje de programación, de arreglos y de ordenación.

Materiales

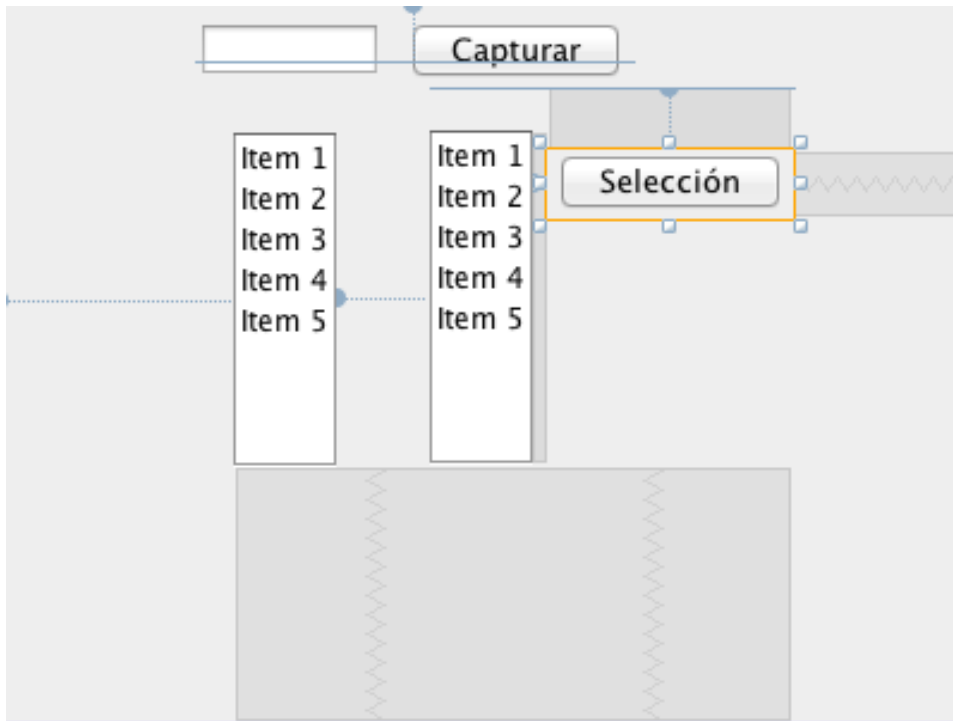
Computadora

Compilador para Java

NetBeans

Procedimiento

- 1.- Se ejecuta el programa NetBeans
- 2.- Se crea un nuevo proyecto cuyo nombre será Selección. Se agregan los siguientes elementos:
un cuadro de texto, 2 listas y 2 botones. Según la figura siguiente:



4. Se codifica la declaración de variables

```
1 /**
   *
   * @author multimediaD
   */
   public class Selección extends javax.swing.JFrame {

1       /**
        * Creates new form Selección
        */
1       public Selección() {
            int A[]=new int[5];
            String AA[]=new String[5];
            int i=0;
            int n=5;

            initComponents();
        }
    }
```

3. se codifica la captura del arreglo dando clic en el botón capturar. Con este botón se captura y al mismo tiempo se muestran los datos introducidos en la primera lista


```

private void capturarActionPerformed(java.awt.event.ActionEvent evt) {
    A[i]=Integer.parseInt(T1.getText());
    AA[i]="" + A[i];
    L1.setListData(AA);
    i++;
    //n=i;
    if(i==5)
        JOptionPane.showMessageDialog(this,"fin de captura");
    else{
        T1.setText("");
        T1.requestFocus();}

    // TODO add your handling code here:
}

```

5. Se codifica el botón de ordenamiento por Selección, dando clic en el botón Selección, el cual mostrará la segunda lista ordenada. Ese botón llamará a la función de Selección (se visualiza más abajo) enviándole como parámetro el arreglo A y recibiendo el arreglo ordenado.

```

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

    int B[]=new int[5];
    A=seleccion(A);
    for (int i=0;i<5;i++)
    {
        AA[i]="" + A[i];
    }
    L2.setListData(AA);

    // TODO add your handling code here:
}

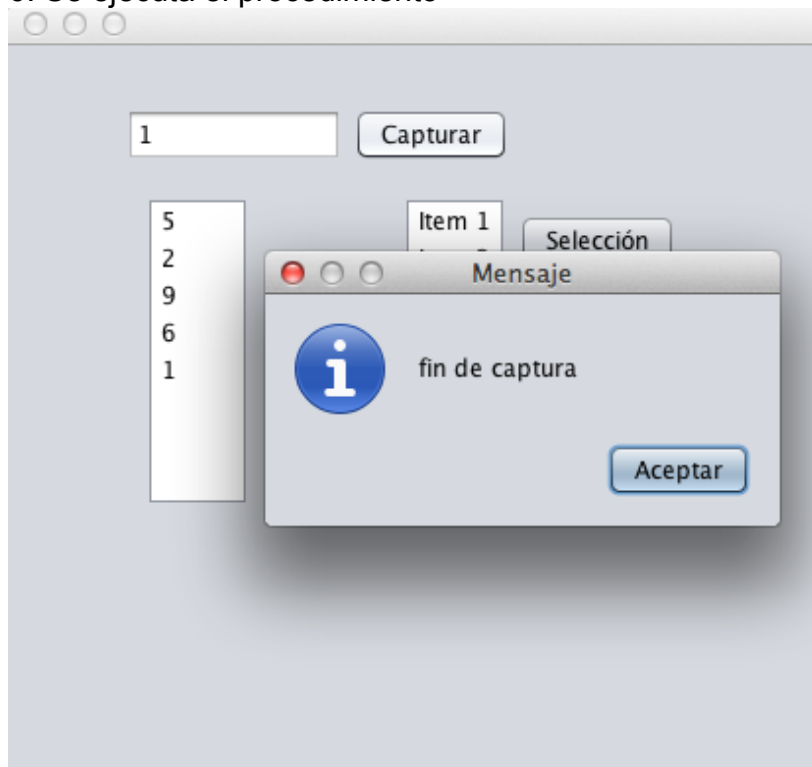
```

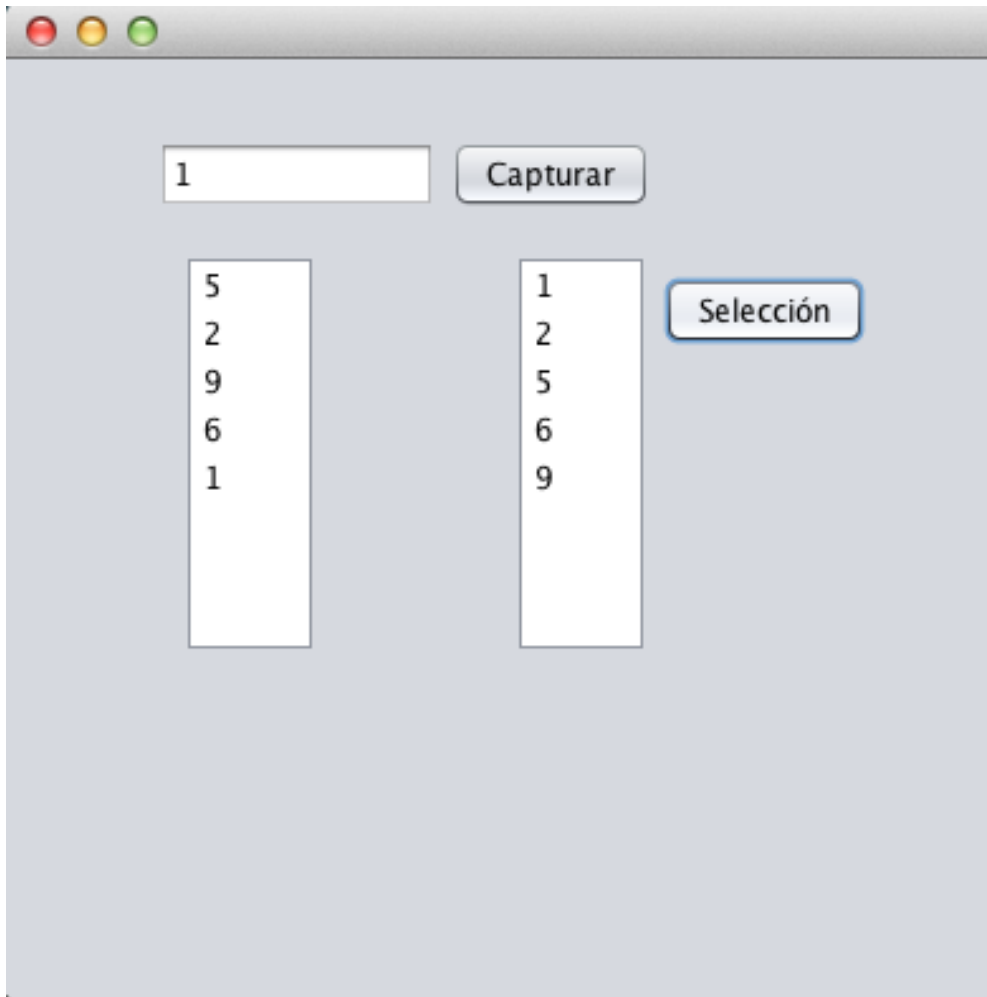
```

public int[] seleccion(int arreglo[])
{
    int contador = 1; // solo util para contar las comparaciones
    for (int i = 0; i < arreglo.length - 1; i++){
        int min = i;
        for (int j = i + 1; j < arreglo.length; j++){
            //System.out.println("Comparacion #" + contador);
            contador++;
            if (arreglo[j] < arreglo[min]){
                min = j;
            }
        }
        if (i != min){
            int aux = arreglo[i];
            arreglo[i] = arreglo[min];
            arreglo[min] = aux;
        }
    }
    return arreglo;
}

```

6. Se ejecuta el procedimiento





7. Aplicando la complejidad algorítmica se observa que contiene 2 bucles y tomando en cuenta que la complejidad lineal crece linealmente con respecto a las iteraciones por lo tanto la complejidad se convierte a $O(n^2)$.

Actividades de Evaluación

1. Escribe un programa que realice capture letras y los ordene en orden de A-Z, anota su complejidad algorítmica (utilice el método de Selección).
2. Escriba la diferencia entre el método de ordenamiento de burbuja y el de selección.
3. Anota el método para que se imprima un arreglo de nombres en orden descendente, es decir de la Z a la A (utilice el método de Selección).

Unidad de competencia II.

Análisis de algoritmos de ordenamiento y búsqueda

Tema 2

Práctica 5: Método de Inserción

Introducción

El ordenamiento por inserción o insertionSort en inglés empieza por un elemento, el cual hace la suposición de que la lista está ordenada, se toma los elementos $k+1$ y se compara con todos los elementos ya ordenados, se detiene cuando se encuentra un elemento menor (todos los elementos mayores han sido desplazados una posición a la derecha) o cuando ya no se encuentran elementos (todos los elementos fueron desplazados).

Descripción de la práctica

Hacer un programa que capture una arreglo, lo ordene por el método de Inserción y lo imprima ordenado y desordenadamente. También se le aplicará la complejidad de Algoritmos

Duración: 2 horas

Objetivos de aprendizaje: EL alumno aprenderá la manera en que funciona el ordenamiento de Inserción

Requisitos Previos

Tener conocimientos sobre algún lenguaje de programación, de arreglos y de ordenación

Materiales

Computadora

Compilador para Java

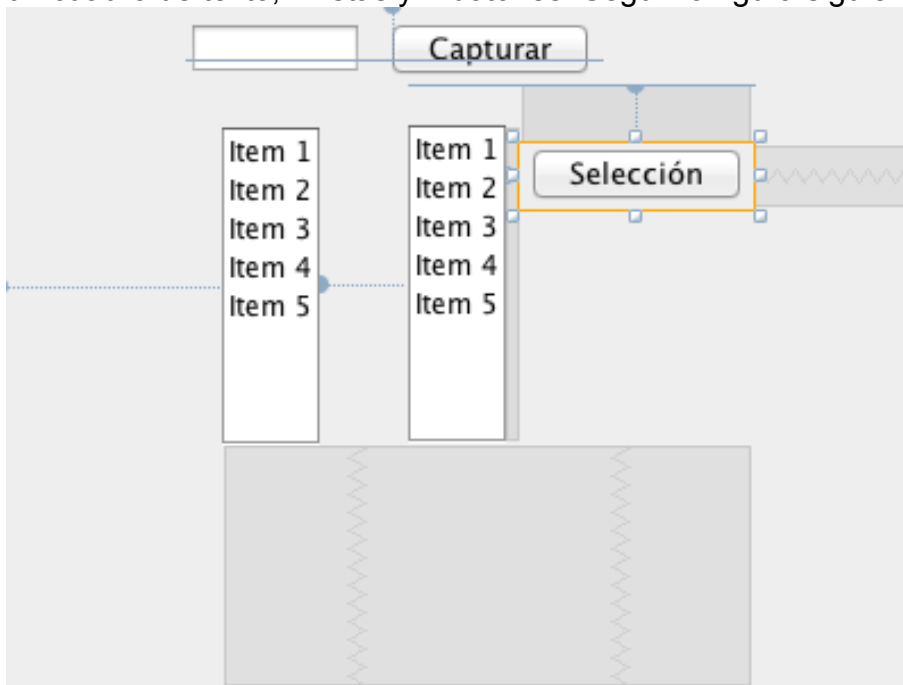
NetBeans

Procedimiento

1.- Se ejecuta el programa NetBeans

2.- Se crea un nuevo proyecto cuyo nombre será Inserción. Se agregan los siguientes elementos:

un cuadro de texto, 2 listas y 2 botones. Según la figura siguiente:



4. Se codifica la declaración de variables

```

3  /**
   *
   * @author multimediaD
   */
public class Inserción extends javax.swing.JFrame {
    int A[]=new int[5];
        String AA[]=new String[5];
        int i=0;
        int n=5;

3      /**
   * Creates new form Inserción
   */
3      public Inserción() {
        initComponents();
    }
}

```

3. se codifica la captura del arreglo dando clic en el botón capturar. Con este botón se captura y al mismo tiempo se muestran los datos introducidos en la primera lista

```

private void capturarActionPerformed(java.awt.event.ActionEvent evt) {
    A[i]=Integer.parseInt(T1.getText());
    AA[i]=""+A[i];
    L1.setListData(AA);
    i++;
    //n=i;
    if(i==5)
        JOptionPane.showMessageDialog(this,"fin de captura");
    else{
        T1.setText("");
        T1.requestFocus();}

    // TODO add your handling code here:
}

```

5. Se codifica el botón de ordenamiento por Inserción, dando clic en el botón Inserción, el cual mostrará la segunda lista ordenada. Ese botón llamará a la función de Inserción (se visualiza más abajo) enviándole como parámetro el arreglo A y recibiendo el arreglo ordenado.

```

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

    int B[]=new int[5];
    A=insercion(A);
    for (int i=0; i<5; i++)
    {
        AA[i]=""+A[i];
    }
    L2.setListData(AA);
}

```

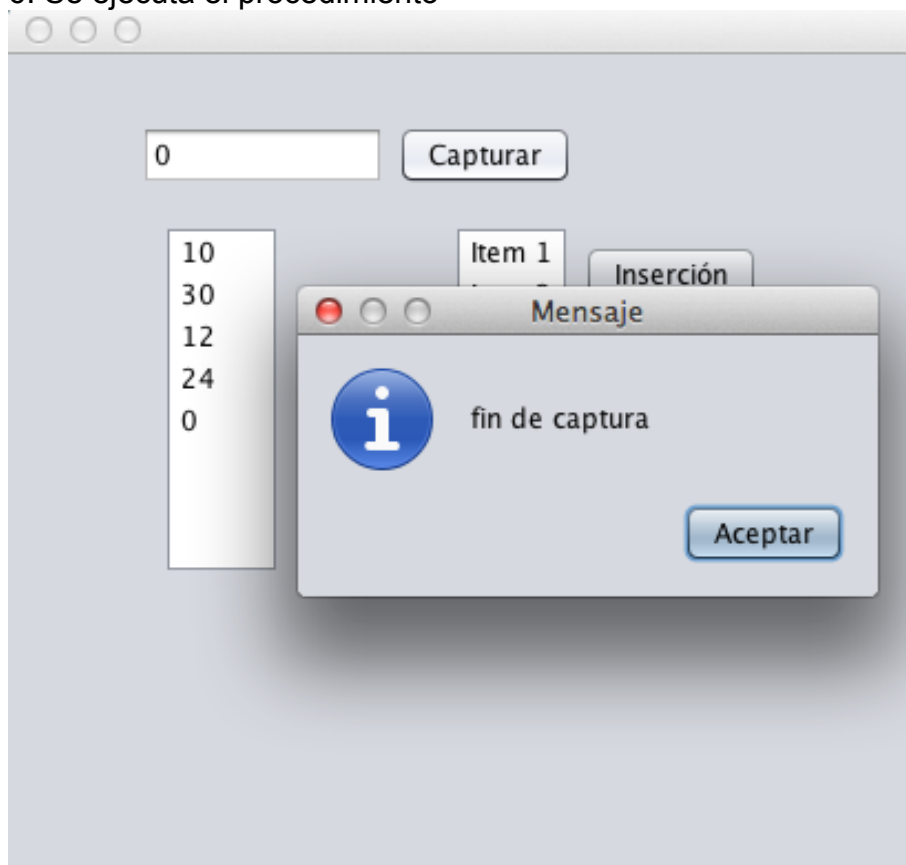
```

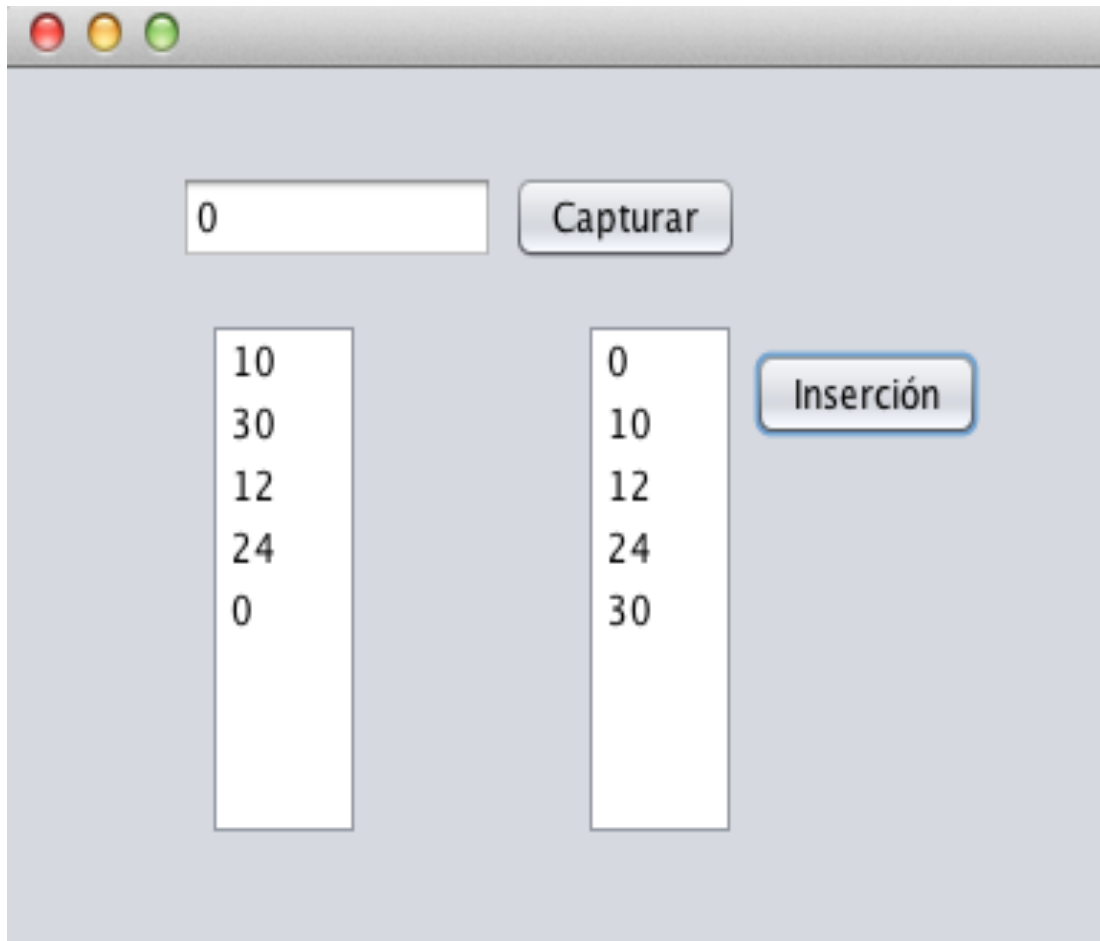
public int[] insercion(int arreglo[])
{
    int p, j;
    int aux;
    for (p = 1; p < A.length; p++){ // desde el segundo elemento hasta
        aux = A[p]; // el final, guardamos el elemento y
        j = p - 1; // empezamos a comprobar con el anterior
        while ((j >= 0) && (aux < A[j])){ // mientras queden posiciones y el
            // valor de aux sea menor que los
            A[j + 1] = A[j]; // de la izquierda, se desplaza a
            j--; // la derecha
        }
        A[j + 1] = aux; // colocamos aux en su sitio
    }

    return arreglo;
}

```

6. Se ejecuta el procedimiento





7. Aplicando la complejidad algorítmica se observa que contiene 2 bucles y tomando en cuenta que la complejidad lineal crece linealmente con respecto a las iteraciones por lo tanto la complejidad se convierte a $O(n^2)$.

Actividades de Evaluación

1. Escribe un programa que realice capture nombres y los ordene en orden de A—Z (utilice el método de Inserción).
2. Anota el método para que se imprima un arreglo de nombres en orden descendente, es decir de la Z a la A (utilice el método de Inserción).
3. Anota su complejidad algorítmica.

Unidad de competencia II.

Análisis de algoritmos de ordenamiento y búsqueda.

Práctica 6: Método de Ordenamiento rápido

Introducción

El **ordenamiento rápido** o **Quicksort**, es un algoritmo que basa su técnica en divide y vencerás, que permite ordenar con menor complejidad.

La manera en que funciona es eligiendo un elemento de la lista de elementos a ordenar, al que le llama pivote. Después resitúa los demás elementos de la lista a cada lado del pivote, así que de un lado quedan todos los menores que el pivote, y al otro los mayores. Los que son iguales es indeterminado en que lado los sitúe.

De esa manera ya se tienen 2 sublistas, Lo mismo hace con las 2 sublistas. Es un procedimiento muy eficiente para grandes cantidades de datos.

Descripción de la práctica

Hacer un programa que capture una arreglo, lo ordene por el método de Quicksort y lo imprima ordenado y desordenadamente. También se le aplicará la complejidad de Algoritmos.

Duración: 2 horas

Objetivos de aprendizaje: EL alumno aprenderá la manera en que funciona el ordenamiento de Quicksort.

Requisitos Previos

Tener conocimientos sobre algún lenguaje de programación, de arreglos y de ordenación.

Materiales

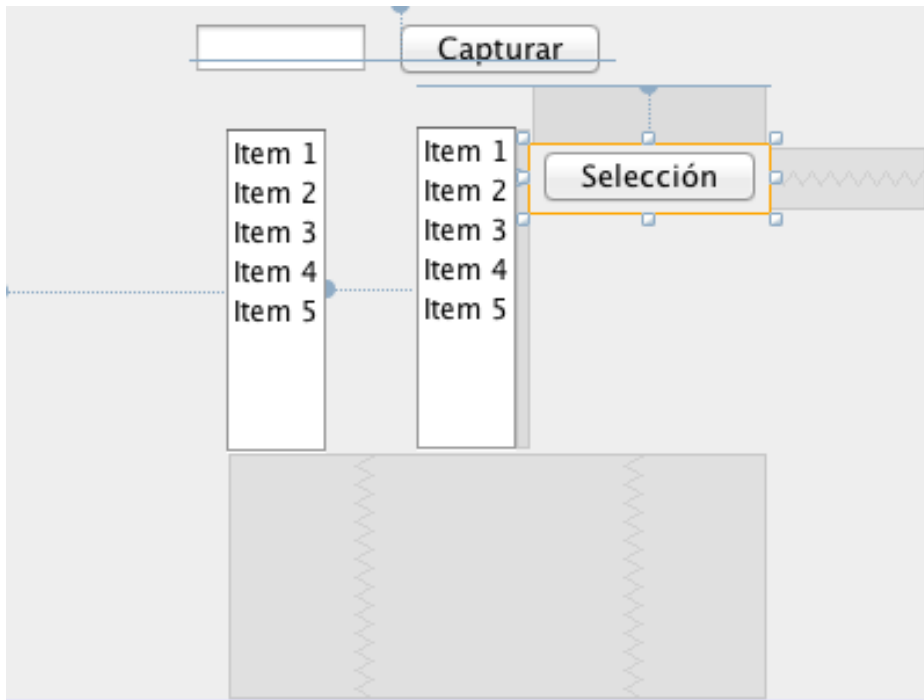
Computadora

Compilador para Java

NetBeans

Procedimiento

- 1.- Se ejecuta el programa NetBeans
- 2.- Se crea un nuevo proyecto cuyo nombre será OrdenRápido. Se agregan los siguientes elementos:
un cuadro de texto, 2 listas y 2 botones. Según la figura siguiente:



4. Se codifica la declaración de variables

```

/**
 *
 * @author multimediaD
 */
public class OrdenRápido extends javax.swing.JFrame {
    int A[]=new int[5];
        String AA[]=new String[5];
        int i=0;
        int n=5;

    /**
     * Creates new form OrdenRápido
     */
    public OrdenRápido() {
        initComponents();
    }
}

```

3. se codifica la captura del arreglo dando clic en el botón capturar. Con este botón se captura y al mismo tiempo se muestran los datos introducidos en la primera lista.

```

private void capturarActionPerformed(java.awt.event.ActionEvent evt) {
    A[i]=Integer.parseInt(T1.getText());
    AA[i]="" + A[i];
    L1.setListData(AA);
    i++;
    //n=i;
    if(i==5)
        JOptionPane.showMessageDialog(this,"fin de captura");
    else{
        T1.setText("");
        T1.requestFocus();}

    // TODO add your handling code here:
}

```

5. Se codifica el botón de ordenamiento por Quicksort, dando clic en el botón Inserción, el cual mostrará la segunda lista ordenada. Ese botón llamará a la función de QuickSort (se visualiza más abajo) enviándole como parámetro el arreglo A y recibiendo el arreglo ordenado.

```

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

    int B[]=new int[5];
    A= QuickSort(A);
    for (int i=0; i<5; i++)
    {
        AA[i]="" + A[i];
    }
    L2.setListData(AA);
}

```

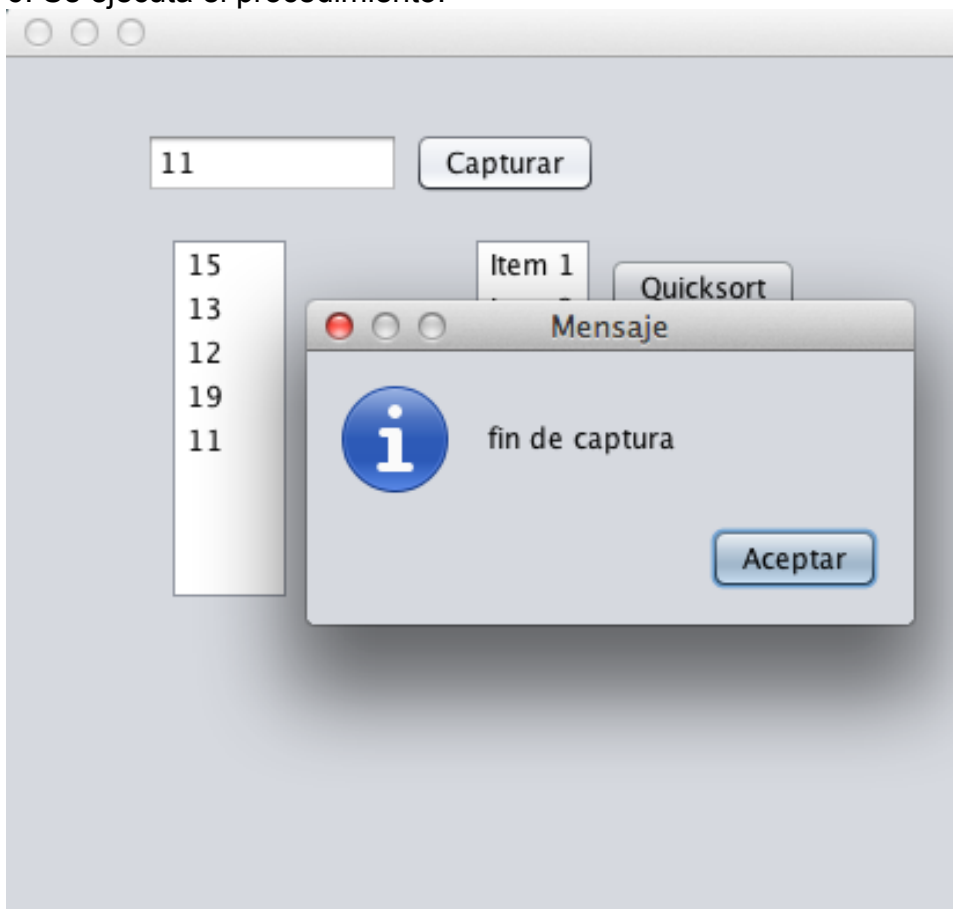
```

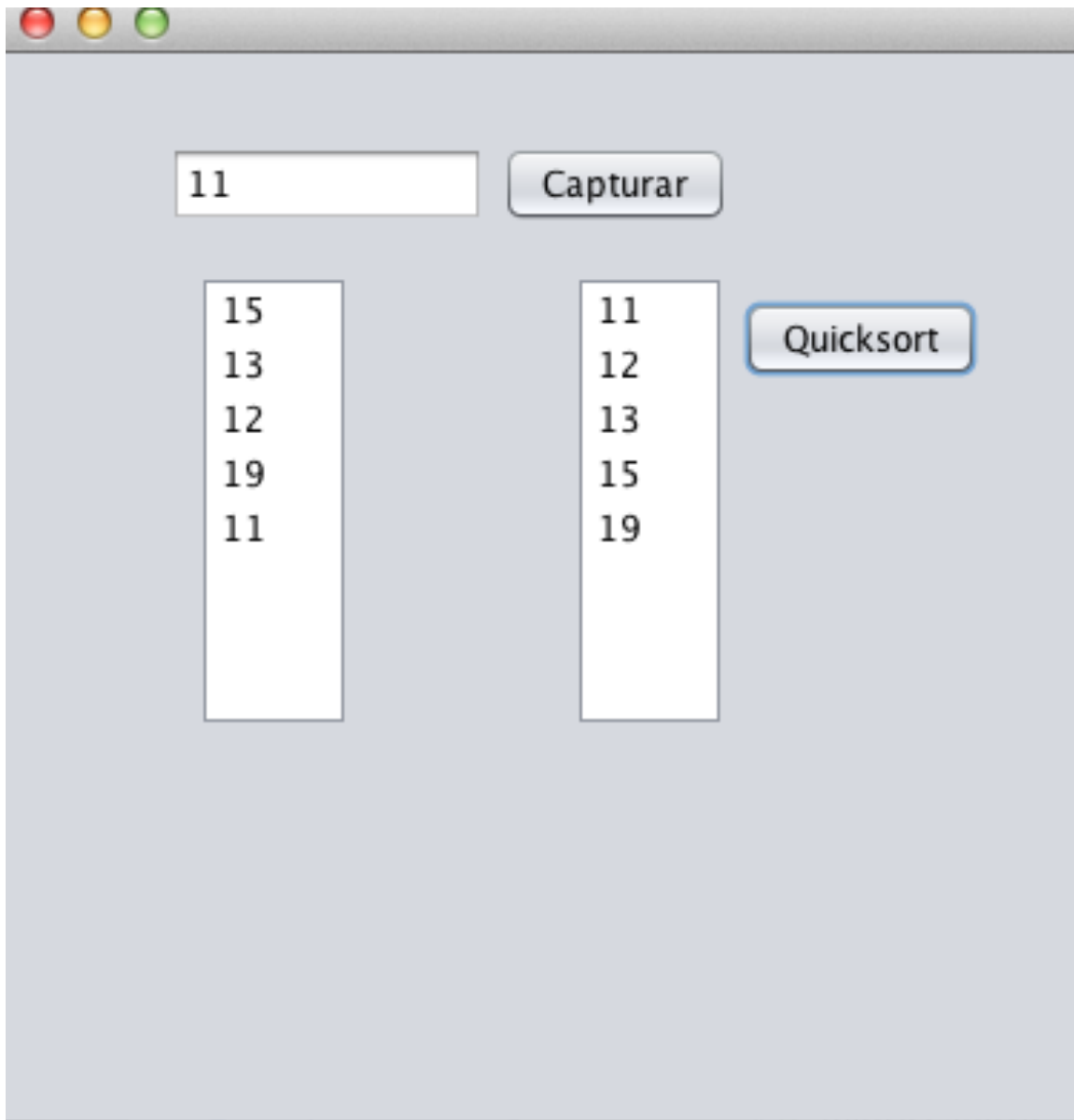
public static void QuickSort(int A[],int izq, int m, int der){
    int i, j, k;
    int [] B = new int[A.length]; //array auxiliar
    for (i=izq; i<=der; i++) //copia ambas mitades en el array auxiliar
        B[i]=A[i];

    i=izq; j=m+1; k=izq;
    while (i<=m && j<=der) //copia el siguiente elemento más grande
        if (B[i]<=B[j])
            A[k++]=B[i++];
        else
            A[k++]=B[j++];
    while (i<=m) //copia los elementos que quedan de la
        A[k++]=B[i++]; //primera mitad (si los hay)
}

```

6. Se ejecuta el procedimiento.





7. Aplicando la complejidad algorítmica se observa que contiene 2 bucles y tomando en cuenta que la complejidad lineal crece linealmente con respecto a las iteraciones por lo tanto la complejidad se convierte a $O(n^2)$

Actividades de Evaluación

1. Escribe un programa que realice capture letras y los ordene en orden de A-Z, anota su complejidad algorítmica (utilice el método de QuickSort).
2. Escriba la diferencia entre el método de ordenamiento de burbuja y el de QuickSort.
3. Anota el método para que se imprima un arreglo de nombres en orden descendente, es decir de la Z a la A (utilice el método de QuickSort).

Unidad de competencia II.

Análisis de algoritmos de ordenamiento y búsqueda

Tema 2

Práctica 7: Método de mezcla

Introducción

El **Mezcla** o **merge sort**, es un algoritmo que basa su técnica en divide y vencerás, que permite ordenar con menor complejidad.

Se Divide la lista desordenada en dos sublistas de aproximadamente la mitad del tamaño, se ordena cada sublista de forma recursiva aplicando el ordenamiento por mezcla.

Descripción de la práctica

Hacer un programa que capture una arreglo, lo ordene por el método de **Mezcla** y lo imprima desordenada y ordenadamente, También se le aplicará la complejidad de Algoritmos.

Duración: 2 horas.

Objetivos de aprendizaje: EL alumno aprenderá la manera en que funciona el ordenamiento de **Mezcla**.

Requisitos Previos

Tener conocimientos sobre algún lenguaje de programación, de arreglos y de ordenación.

Materiales

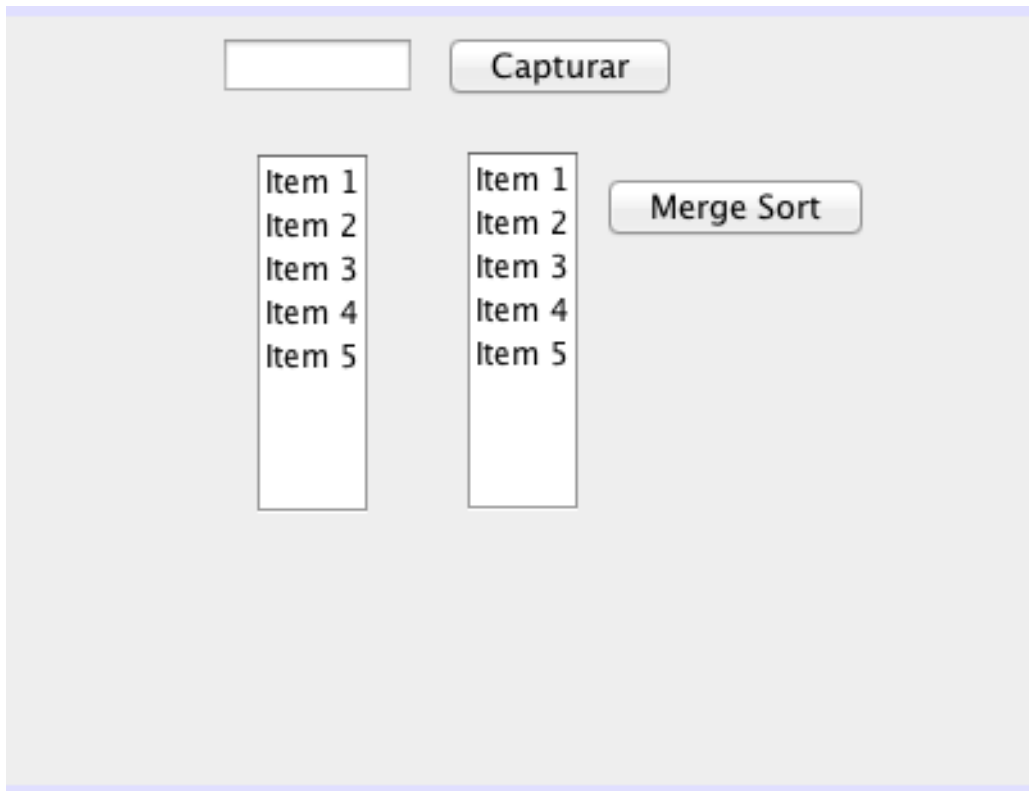
Computadora

Compilador para Java

NetBeans

Procedimiento

- 1.- Se ejecuta el programa NetBeans
- 2.- Se crea un nuevo proyecto cuyo nombre será Mezcla. Se agregan los siguientes elementos:
un cuadro de texto, 2 listas y 2 botones. Según la figura siguiente:



4. Se codifica la declaración de variables.

```
public class Mezcla extends javax.swing.JFrame {
    int A[]=new int[5];
        String AA[]=new String[5];
        int i=0;
        int n=5;
    /**
     * Creates new form Mezcla
     */
    public Mezcla() {
        initComponents();
    }
}
```

3. se codifica la captura del arreglo dando clic en el botón capturar. Con este botón se captura y al mismo tiempo se muestran los datos introducidos en la primera lista

```

private void capturarActionPerformed(java.awt.event.ActionEvent evt) {
    A[i]=Integer.parseInt(T1.getText());
    AA[i]=""+A[i];
    L1.setListData(AA);
    i++;
    //n=i;
    if(i==5)
        JOptionPane.showMessageDialog(this,"fin de captura");
    else{
        T1.setText("");
        T1.requestFocus();}

    // TODO add your handling code here:
}

```

5. Se codifica el botón de ordenamiento por MergeSort, dando clic en el botón Merge Sort, el cual mostrará la segunda lista ordenada. Ese botón llamará a la función de MergeSort (se visualiza más abajo) enviándole como parámetro el arreglo A y recibiendo el arreglo ordenado.

```

private void MergeSortActionPerformed(java.awt.event.ActionEvent evt) {
    int B[]=new int[5];
    mergesort(A,2,3);
    for (int i=0;i<5;i++)
    {
        AA[i]=""+A[i];
    }
    L2.setListData(AA);

    // TODO add your handling code here:
}

```

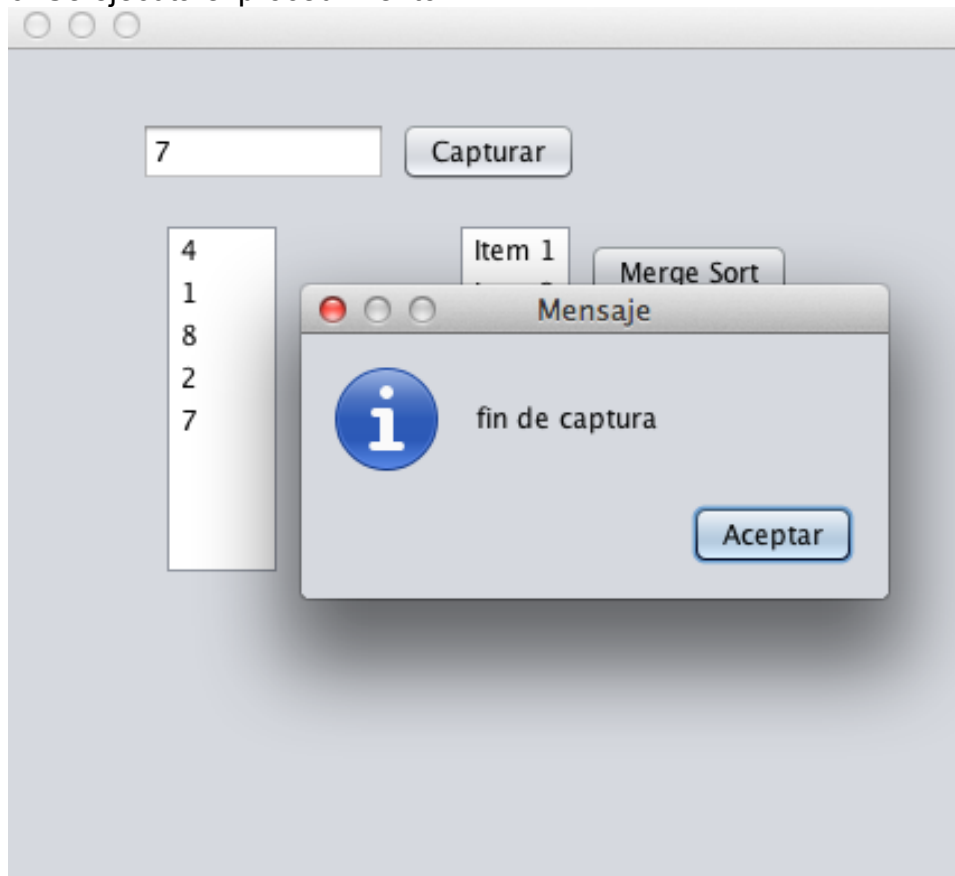


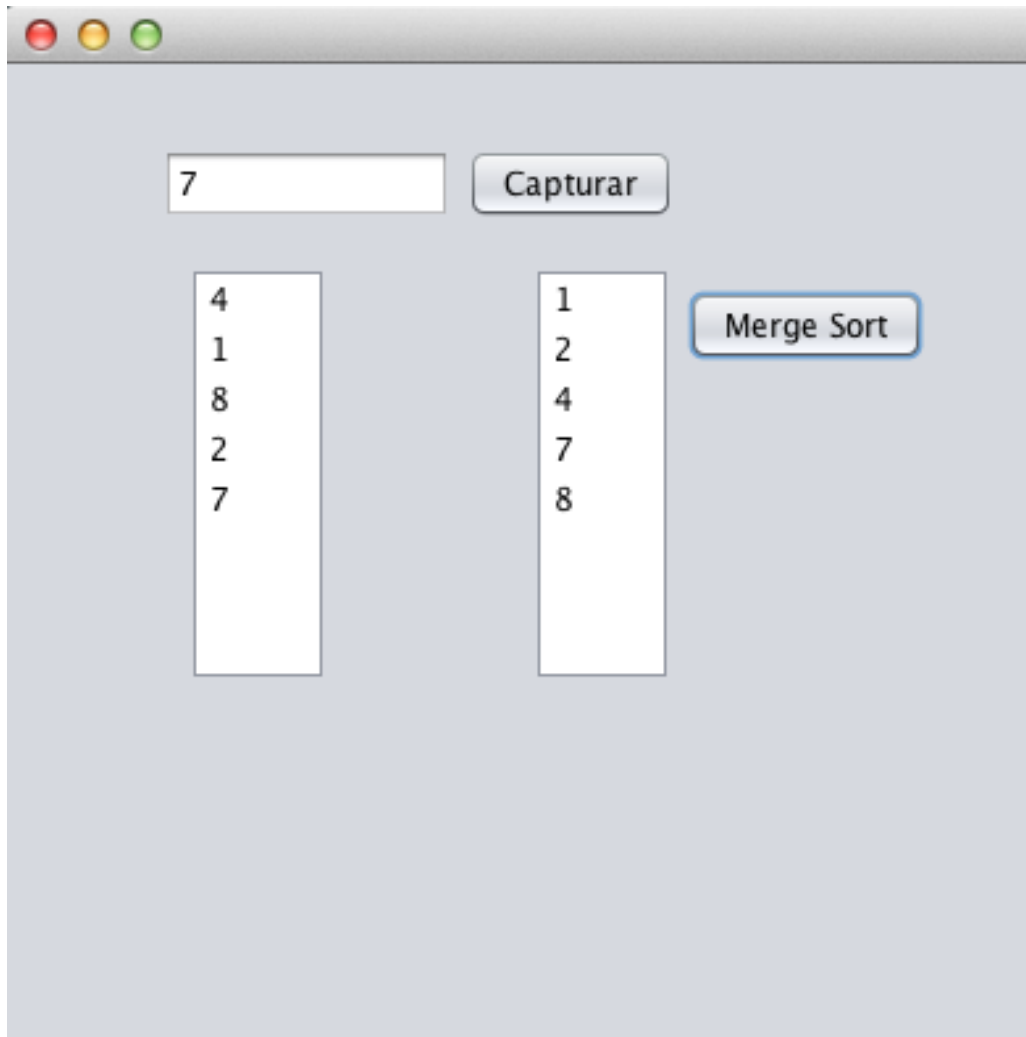
```

public static int[] mergesort(int A[],int izq, int der){
    if (izq<der){
        int m=(izq+der)/2;
        mergesort(A,izq, m);
        mergesort(A,m+1, der);
        merge(A,izq, m, der);
    }
    return A;
}

```

6. Se ejecuta el procedimiento





7. Aplicando la complejidad algorítmica se observa que contiene 2 bucles y tomando en cuenta que la complejidad de los algoritmos que implican bucles con menos iteraciones es del orden algorítmica, ésta sería $O(n \log n)$.

Actividades de Evaluación

1. Escribe un programa que realice capture letras y los ordene en orden de A-Z, anota su complejidad algorítmica (utilice el método de MergeSort).
2. Escriba la diferencia entre el método de ordenamiento de QuickSort y el de MergeSort.
3. Anota el método para que se imprima un arreglo de nombres en orden descendente, es decir de la Z a la A (utilice el método de QuickSort).

Unidad de competencia II.

Análisis de algoritmos de ordenamiento y búsqueda

Práctica 8: Búsqueda Secuencial

Introducción

Se diseña para localizar un elemento con ciertas propiedades dentro de un vector. Su funcionamiento es hacer repetidas comparaciones del elemento buscado con los elementos del vector.

Descripción de la práctica

Hacer un programa que capture una arreglo, solicite un número a buscar en la lista e imprima si se encuentra o no y la posición en la que se encuentra.

Duración: 2 horas

Objetivos de aprendizaje: EL alumno aprenderá la manera en que funciona la búsqueda secuencial.

Requisitos Previos

Tener conocimientos sobre algún lenguaje de programación, de arreglos y de búsquedas

Materiales

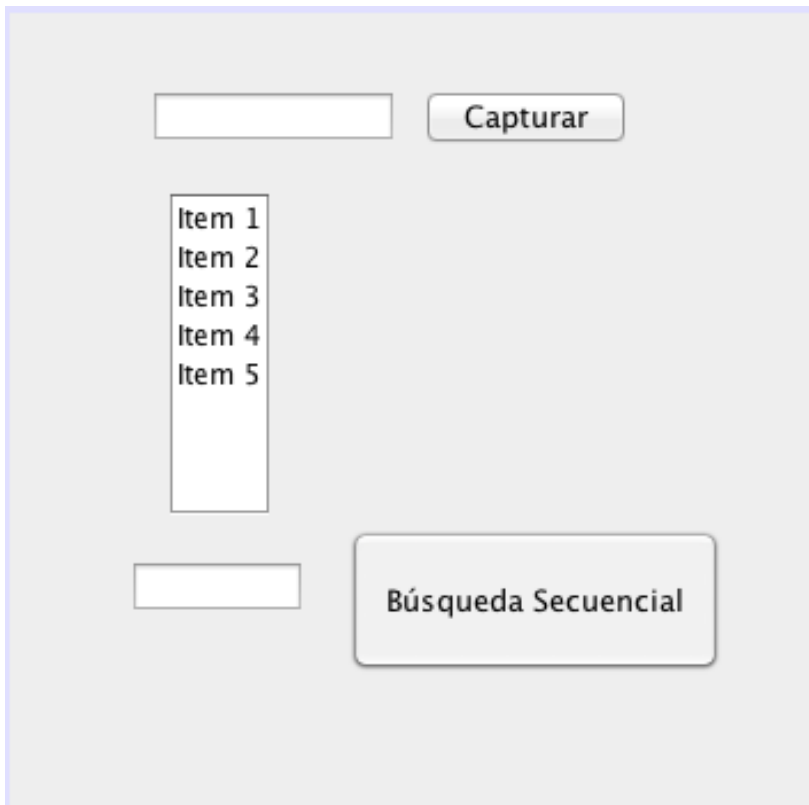
Computadora

Compilador para Java

NetBeans

Procedimiento

- 1.- Se ejecuta el programa NetBeans
- 2.- Se crea un nuevo proyecto cuyo nombre será BúsquedaSecuencia. Se agregan los siguientes elementos:
Dos cuadros de texto, 1 listas y 2 botones. Según la figura siguiente:



4. Se codifica la declaración de variables

```
public class BúsquedaSecuencial extends javax.swing.JFrame {
    int A[]=new int[5];
        String AA[]=new String[5];
        int i=0;
        int n=5;
}
/**
 * Creates new form BúsquedaSecuencial
 */
public BúsquedaSecuencial() {
    initComponents();
}
```

3. se codifica la captura del arreglo dando clic en el botón capturar. Con este botón se captura y al mismo tiempo se muestran los datos introducidos en la primera lista

```

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {

    int B[]=new int[5];
    int num1=Integer.parseInt(T2.getText());
    int numero=BSecuencial(A,num1);
    if(numero !=-1 )
        JOptionPane.showMessageDialog(this,"el número "+ num1 + " se encuentra en la posición "+ numero);
    else
        JOptionPane.showMessageDialog(this,"el número "+ num1 + " no se encuentra en la lista");

    // TODO add your handling code here:
}

```

5. Se codifica el botón de búsqueda secuencial, dando clic en el botón BSecuencial, el cual mostrará la segunda lista ordenada. Ese botón llamará a la función de BSecuencial (se visualiza más abajo) enviándole como parámetro el arreglo A y recibiendo el arreglo ordenado.

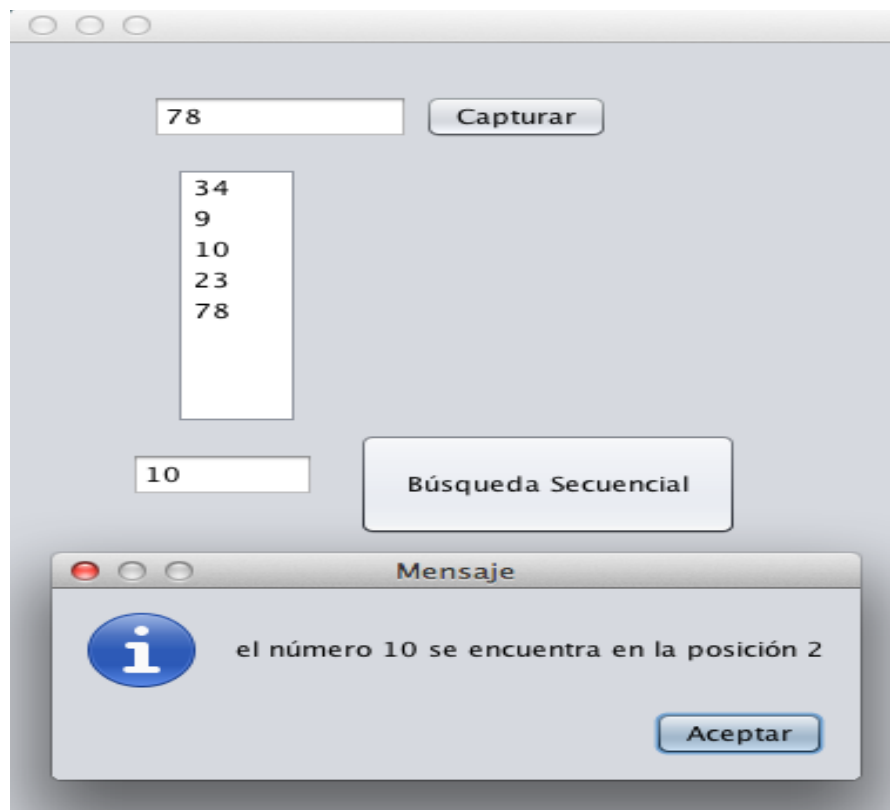
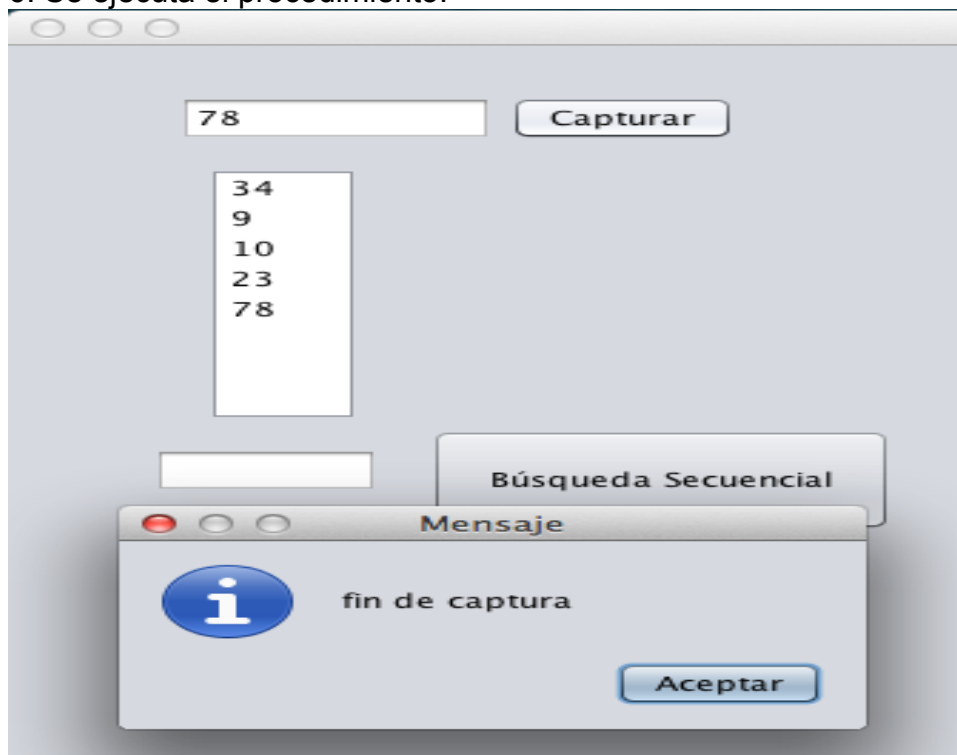
```

public int BSecuencial(int arreglo[],int nB)
{
    int i;
    int n=-1;
    for( i=0;i<=arreglo.length;i++)
    {
        if(arreglo[i]== nB)
        {
            n=i;
            break;
        }
    }

    return n;
}

```

6. Se ejecuta el procedimiento.



7. Aplicando la complejidad algorítmica se observa que contiene 1 bucle y tomando en cuenta que la complejidad lineal crece linealmente con respecto a las iteraciones la complejidad se convierte a $O(n)$.

Actividades de Evaluación

1. Anote cual sería el mejor y el peor de los casos en el algoritmo de búsqueda secuencial.
2. Anota las el procedimiento de búsqueda secuencial utilizando un bucle while.

Unidad de competencia II.

Análisis de algoritmos de ordenamiento y búsqueda

Práctica 8: Búsqueda Binaria

Introducción

búsqueda binaria, también conocida como búsqueda de intervalo medio o búsqueda logarítmica, es un algoritmo que busca y encuentra la posición de un elemento en un arreglo. Lo hace comparando el elemento con el elemento medio del arreglo, si no lo encuentra elimina esa mitad de arreglo y vuelve hacer sub listas de la parte del arreglo que quedó, repite este procedimiento hasta localizar el elemento buscado.

Descripción de la práctica

Hacer un programa que capture una arreglo, solicite un número a buscar en la lista e imprima si se encuentra o no y la posición en la que se encuentra

Duración: 2 horas

Objetivos de aprendizaje: EL alumno aprenderá la manera en que funciona la búsqueda Binaria.

Requisitos Previos

Tener conocimientos sobre algún lenguaje de programación, de arreglos y de búsquedas

Materiales

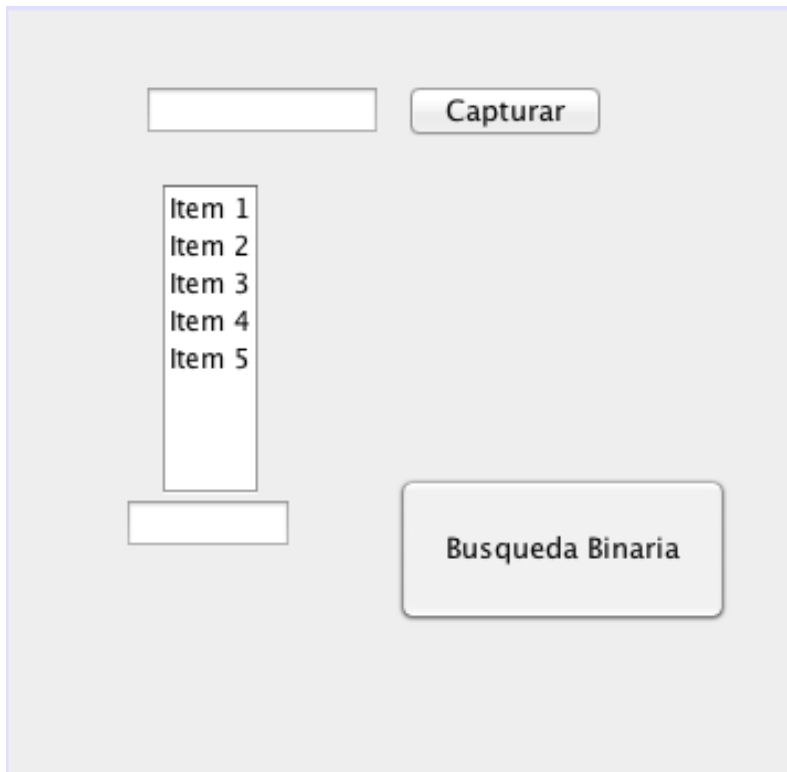
Computadora

Compilador para Java

NetBeans

Procedimiento

- 1.- Se ejecuta el programa NetBeans
- 2.- Se crea un nuevo proyecto cuyo nombre será BúsquedaBinaria. Se agregan los siguientes elementos:
Dos cuadros de texto, 1 listas y 2 botones. Según la figura siguiente:



4. Se codifica la declaración de variables

```
public class BúsquedaBinaria extends javax.swing.JFrame {
    int A[]=new int[5];
        String AA[]=new String[5];
        int i=0;
        int n=5;|
}
/**
 * Creates new form BúsquedaBinaria
 */
public BúsquedaBinaria() {
    initComponents();
}
```

3. se codifica la captura del arreglo dando clic en el botón capturar. Con este botón se captura y al mismo tiempo se muestran los datos introducidos en la primera lista.

```

private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {

    int B[]=new int[5];
    int num1=Integer.parseInt(T2.getText());
    int numero=busquedaBinaria(A,num1);
    if(numero !=-1 )
        JOptionPane.showMessageDialog(this,"el número "+ num1 + " se encuentra en la posición "+ numero);
    else
        JOptionPane.showMessageDialog(this,"el número "+ num1 + " no se encuentra en la lista");

    // TODO add your handling code here:
}

```

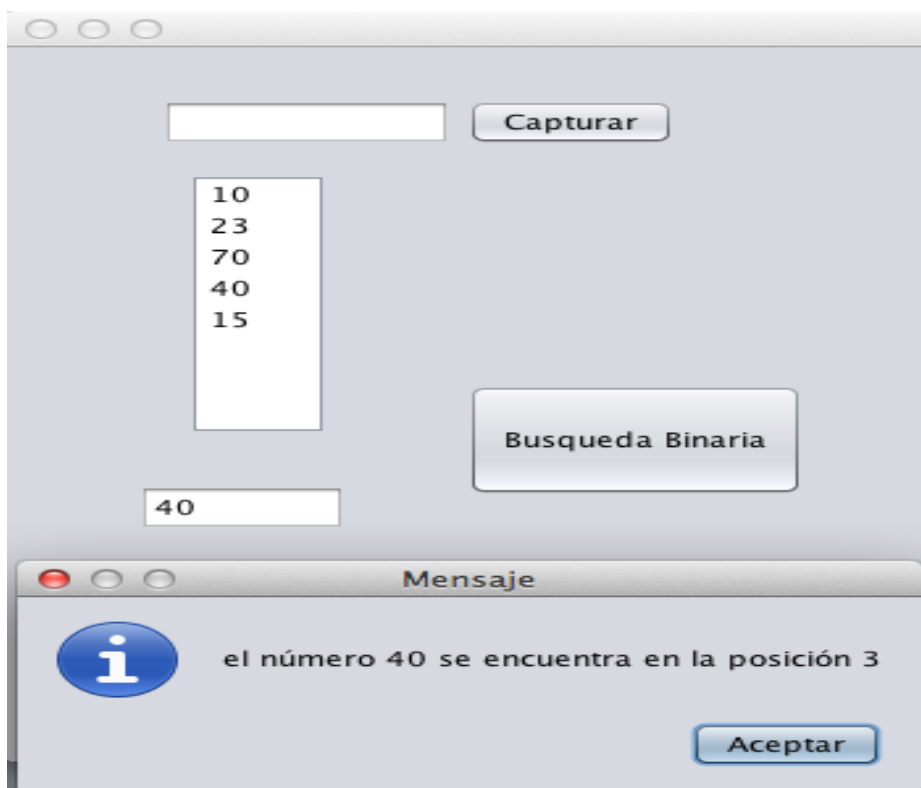
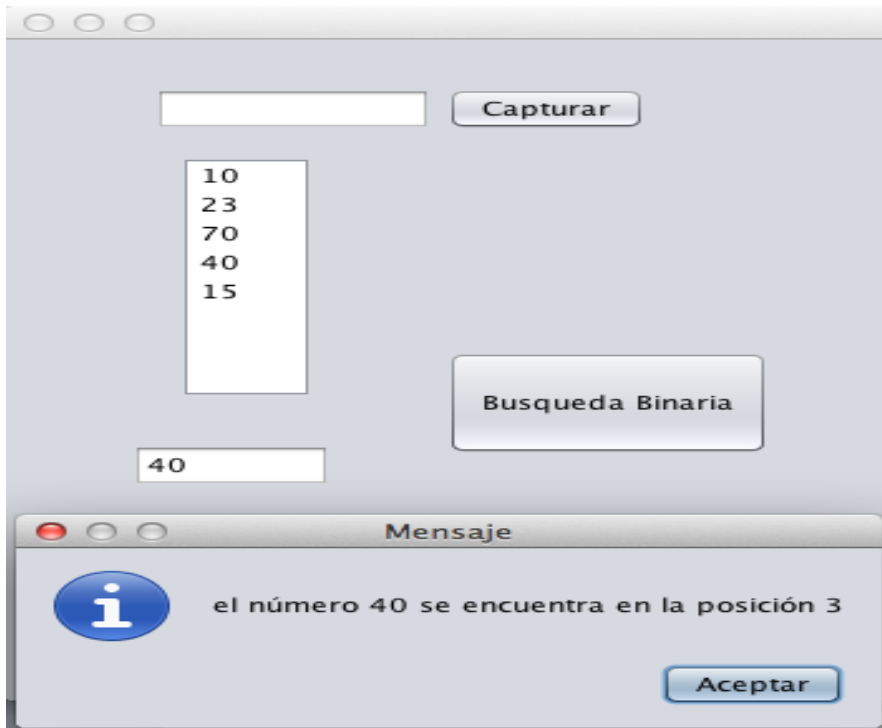
5. Se codifica el botón de ordenamiento por Búsqueda Binaria, dando clic en el botón Inserción, el cual mostrará la segunda lista ordenada. Ese botón llamará a la función de busquedaBinaria (se visualiza más abajo) enviándole como parámetro el arreglo A y recibiendo el arreglo ordenado

```

public static int busquedaBinaria(int vector[], int dato){
    int n = vector.length;
    int centro,inf=0,sup=n-1;
    while(inf<=sup){
        centro=(sup+inf)/2;
        if(vector[centro]==dato) return centro;
        else if(dato < vector [centro] ){
            sup=centro-1;
        }
        else {

```

6. Se ejecuta el procedimiento.



7. Aplicando la complejidad algorítmica se observa que contiene 2 bucles y tomando en cuenta que la complejidad de los algoritmos que implican bucles con menos iteraciones (en este caso, va dividiendo y buscando en la mitad correspondiente) es del orden algorítmica, ésta sería $O(n \log n)$.

Actividades de Evaluación

1. Construir un programa que permita cargar los nombres de 20 alumnos y sus notas respectivas (2 vectores). Buscar el nombre de un alumno e imprimir el nombre del alumno y su correspondiente calificación, es decir, imprimir la el índice del nombre localizado pero del segundo arreglo.

Referencias

1. Yakov Fain, Programación Java /Editorial Anaya 2010.
2. Enrique Gómez Jiménez.
3. Desarrollo de software NetBeans 7.1, Programe para Escritorio, Web y Dispositivos Móviles, Editorial Ra-Ma 2012.
4. PJ Deitel HM Deitel, Java como programar, Editorial Pearson / Educación 2013
5. Luis Joyanes, Programación en Java 6, McGraw-Hill Interamericana 2011